A PARALLEL APPROACH FOR BACKPROPAGATION LEARNING OF NEURAL NETWORKS

CRESPO, M., PICCOLI F.,PRINTISTA M.[1],
GALLARD R.[2],

Grupo de Interés en Sistemas de Computación[3]
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950 - Local 106
5700 - San Luis
Argentina
E-mail:{mcrespo,mpiccoli,mprinti,rgallard}@inter2.unsl.edu.ar
{mcrespo,mpiccoli,mprinti,rgallard}@unsl.edu.ar
Phone: +54 652 20823
Fax : +54 652 30224

ABSTRACT

Learning algorithms for neural networks involve CPU intensive processing and consequently great effort has been done to develop parallel implemetations intended for a reduction of learning time.
This work briefly describes parallel schemes for a backpropagation algorithm and proposes a distributed system architecture for developing parallel training with a partition pattern scheme. Under this approach, weight changes are computed concurrently, exchanged between system components and adjusted accordingly until the whole parallel learning process is completed. Some comparative results are also shown.

KEYWORDS: Neutral networks, parallelised backpropagation, partitioning schemes, pattern partitioning, system architecture.

[1]Members of the Research Group.

[2]Full Professor at the Informatics Department. Head of the Research Group.

[3]The Research Group is supported by the Universidad Nacional de San Luis and the CONICET (Science and Technology Research Council).

1. INTRODUCTION

Neural nets learn by training, not by being programmed. Learning is the process of adjustment of the neural network to external stimuli. After learning it is expected that the network will show *recall* and *generalization* abilities. By recall we mean the capability to recognise inputs from the training set, that is to say, those patterns presented to the network during the learning process. By generalization we mean the ability to produce reasonable outputs associated with new inputs of the same total pattern space. These properties are attained during the slow process of learning. Many approaches to speedup the training process has been devised by means of parallelism.
The backpropagation algorithm (BP) is one of the most popular learning algorithms and many approaches to parallel implementations has been studied [7], [12], [13].
To parallelise BP either the network or the training pattern space is partitioned. In *network partitioning*, the nodes and weights of the neural network are distributed among diverse processors. Hence the computations due to node activations, node errors and weight changes are parallelised. In *pattern partitioning* the whole neural net is replicated in different processors and the weight changes due to distinct training patterns are parallelised.
This paper shows the design of a distributed support for parallel learning of neural networks using a pattern partitioning approach. Results on speedup in learning and its impact on recall and generalization are shown.

2. BACKPROPAGATION NETWORKS

A backpropagation neural network is composed of at least three unit layers; an input, an output and one ore more hidden (intermediate) layers. Figure 1 shows a three layer BP network.

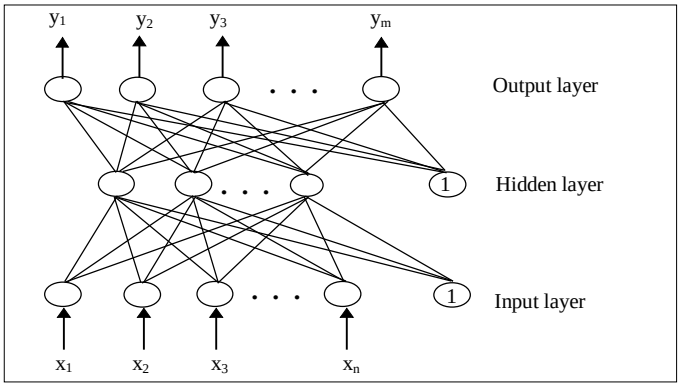

Fig. 1 - A Backpropagation Network

Given a set of p 2-tuples of vectors (x_1 y_1), (x_2 y_2), ..., (x_p y_p), which are samples of a functional mapping $y = \phi(x) : x \in R^N$, $y \in R^M$, the goal is to train the network in order it learns an approximation $o = y' = \phi'(x)$. The learning process is carried out by using a two phase cycle; *propagation* or *forward phase* and *adaptation* or *backward phase* [6], [11], [12].
Once an input pattern is applied as an excitation to the input layer, it is propagated throughout the remaining layers up to the output layer where the current output of the network is generated. This output is contrasted against the desired output and an error value is computed as a function of the outcome error in each output unit. This makes up the *forward phase*.
The learning process include adjusting of weights (adaptation) in the network in order

to minimise the error function. For this reason, the error obtained is propagated back from each node of the output layer to the corresponding (contributors) nodes of the intermediate layer. However each of these intermediate units receives only a portion of the total error according to its relative contribution to the current output. This process is repeated from one layer to the previous one until each node in the network receives an error signal describing its relative contribution to the total error. Based on this signal, weights are corrected in a proportion directly related to the error in the connected units. This makes up the *backward phase*.
During this process, as the training is progressing, nodes in the intermediate levels are organised in such a way that different nodes recognise different features of the total training space. After training, when a new input pattern is supplied, units in the hidden layers will generate active outputs if such an input pattern preserves the features they (individually) learnt. Conversely, if such an input pattern do not contain those known features then these units will be inclined to inhibit their outputs.
It has been shown that during training, backpropagation neural nets tend to develop internal relationships between units, as to categorise training data into pattern classes. This association can be either evident or not to the observer. The point here is that, firstly, the net finds an internal representation which enables it to generate appropriate responses when those patterns used during training are subsequently submitted to the network and secondly, the network will classify those patterns never seen before according to the features they share (resemblance) with the training patterns.

3. PATTERN PARTITIONING FOR PARALLEL BACKPROPAGATION

Pattern partitioning replicates the neural net structure (units, edges and associated weights) at each processor and the training set is equally distributed among processors. Each processor performs the propagation and the adaptation phases for the local set of patterns. Also, each processor accumulates the weight changes produced by the local patterns which afterward are exchanged with other processors to update weight values.
This scheme is suitable for problems with a large set of training patterns and fit properly to run on local memory architectures [1], [8], [10].
Different training techniques, or regimes, can be implemented in a backpropagation algorithm to implement the learning process [6], [11], [12]. In this work, because it is appropriated for a distributed environment, the choice was the *set-training* regime. Under this technique weight changes are accumulated for all training patterns before updating any of them, with a subsequent *update all* stage each time all patterns in the training set were presented.

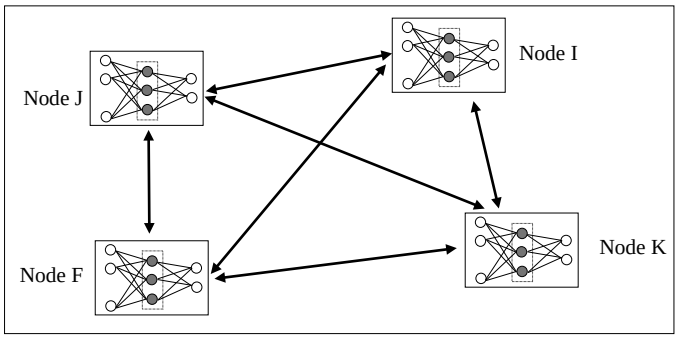

Fig. 2. A Pattern Partitioning Scheme

4. THE PARALLEL LEARNING ALGORITHM

To implement a pattern partitioning scheme, the whole neural network is replicated among P processors and each processor carries out the learning process using Ts/P patterns where Ts is the size of the training set. As shown in figure 2, weight changes are performed in parallel and then the corresponding accumulated weight changes vectors, *awc*, are exchanged between processors. Now we describe the basic steps of a backpropagation learning algorithm under the *set-training* regime, also known as *per-epoch training*[4]. The superindexes *h* and *o* identify hidden and output units respectively.

Batch-Training Algorithm

Repeat

1. For each pattern $x_p = (x_{p1}, x_{p2}, ..., x_{pN})$

1.1 Compute the output of units in the hidden layer:

$$net_{pj}^{h} = \sum_{i=1}^{N} w_{ji}^{h} x_{pi} + \Theta_{j}^{h}$$

$$i_{pj} = f(net_{pj}^{h})$$

1.2 Compute the output of units in the output:

$$net_{pk}^{o} = \sum_{j=1}^{L} w_{kj}^{o} i_{pj} + \Theta_{k}^{o}$$

$$o_{pk} = f(net_{pk}^{o})$$

1.3 Compute error terms for the units in the output layer:

$$\delta_{pk}^{o} = (y_{pk} - o_{pk})\, f'(net_{pk}^{o})$$

1.4 Compute error terms for the units in the hidden layer:

$$\delta_{pj}^{h} = f'(net_{pj}^{h}) \sum_{k} \delta_{pk}^{o} w_{kj}^{o}$$

1.5 Compute weight changes in the output layer:

$$c_{kj}^{o} = c_{kj}^{o} + \eta\, \delta_{pk}^{o} i_{pj}$$

1.6 Compute weight changes in the hidden layer:

$$c_{ji}^{h} = c_{ji}^{h} + \eta\, \delta_{pj}^{h} x_{i}$$

2. Send local c^h and c^o and Receive remote c^h y c^o.

3. Update weights changes in the output layer:

$$w_{kj}^{o} = w_{kj}^{o} + (c_{kj}^{o}\ (local) + c_{kj}^{o}\ (remote))$$

4. Update weights changes in the hidden layer:

$$w_{ji}^{h} = w_{ji}^{h} + (c_{ji}^{h}\ (local) + c_{ji}^{h}\ (remote))$$

Until $\left(E = \sum_{p=1}^{T} \left(\frac{1}{2} \sum_{k=1}^{M} (y_{pk} - o_{pk})^2\right) < \text{maximum accepted error}\right)$ or

(number of iterations < maximum number of iterations)

The subindexes *p*, *i*, *j* and *k* identify the p[th] input pattern, the i[th] input unit, the j[th] hidden unit and the k[th] output unit respectively. w_{ij} is the weight corresponding to the connection between unit *j* and unit *i*, c_{ij} is the accumulated change for the weight

[4] During an epoch, or batch, the submission of all patterns in the partition, the corresponding computations and the accumulation of weight changes must be performed before weights update takes place, then the next epoch begin.

corresponding to the connection between unit *j* and unit *i,* Θ is the bias and *N*, *L* and *M* identify the number of input, hidden and output units respectively. *f* denotes the sigmoid function which is used to compute the activation of each node.

5. PARALLEL LEARNING SUPPORT

In this section we present a brief description of the underlying system architecture and procedures supporting the parallel learning process, running on the processors distributed in a LAN of workstations [3]. Each processor is allocated for a replicated neural network. The parallel learning support presented here is independent of the neural network structure.

5.1 System Architecture

Figure 3 shows the support system architecture, processes and interactions. The parallel learning algorithm is independently initiated by a process at each LAN node. The *ProcW* process, will be responsible of local learning processing and when needed will request a service to exchange vectors *awc. ProcW* forks twice to create child processes *HI* and *HJ* for communication with other LAN nodes.

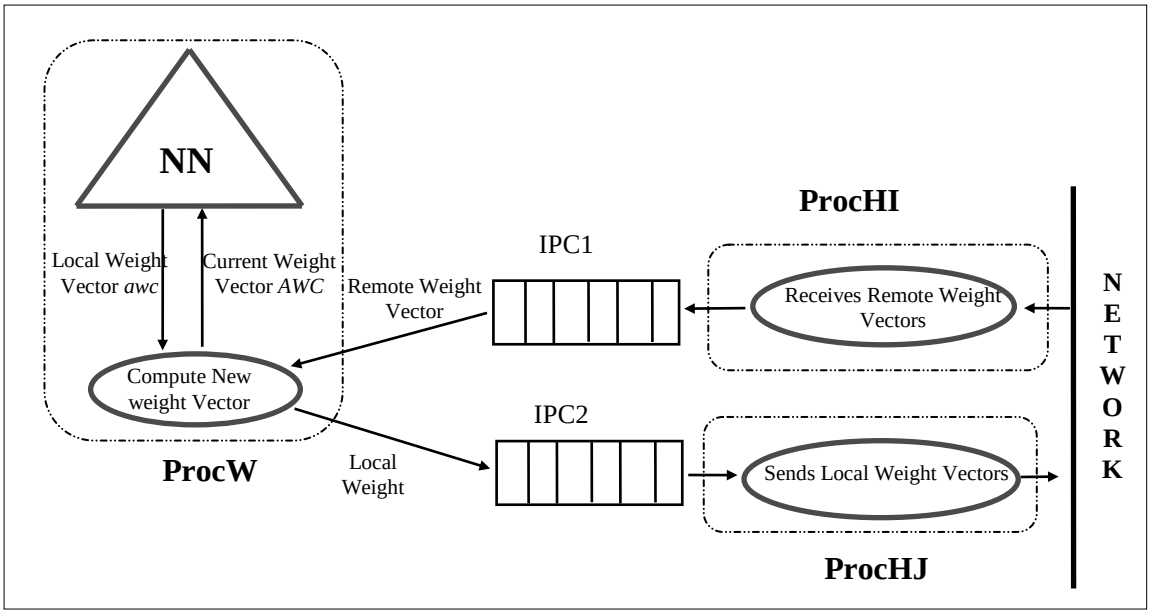

Fig. 3. - Support System Architecture

The tasks performed by each process are:

- Main process *ProcW* runs the learning algorithm for the neural net NN. After each epoch *ProcW* request the following services:
  Sending of the local *awc* vector.
  Receiving of remote *awc* vectors.
  To update the weights for every pattern in the partition it creates a current *AWC* vector by gathering the local and remote *awc* vectors.

- Process *ProcHI* receives the *awc* vectors sent by remote *ProcHJ* processes.
- Process *ProcHJ* sends the local *awc* vectors generated during one ore more epochs to the remote *ProcHI* processes.

These three processes execute independently but they communicate for exchanging information. In order to allow the concurrent execution of learning and the external communication with remote processes, a set of *non blocking* mechanisms where used [4], [9], [14], [15].
This communication is handled via interprocess communication mechanisms (IPCs).
The interaction between processes *ProcW* and *ProcHI* is established via IPC1 when after an epoch the learning process request to distribute its *awc* vector.
The interaction between processes *ProcHJ* and *ProcW* is established via IPC2 when after an epoch the learning process request remote *awc* vectors to subsequently determine the current *AWC* vector.

6. THE PARALLEL VERSION OF THE NND SUPERVISOR

In our research, Neural Network Devices (NND) are used to supervise load distribution in Distributed Systems. In particular, for parallelising BP, the test case was chosen as a reduced version of the one discussed elsewhere [2]. The neural network was built as a Feed Forward Neural Net (FFNN) of 14 input nodes, 2 output nodes and one hidden layer of 30 nodes with a total number of 527 interconnections.
The pattern space contains a total number of 2,376 patterns. For the experiments discussed later only one third (about 800 samples) of the pattern space was enough to obtain satisfactory results.
In previous works by creating prototypes we corroborate that claimed neural network features such as fast response, storage efficiency, fault tolerance and graceful degradation in face of scarce or spurious inputs make them appropriate tools for Intelligent Computer Systems.
Now, parallel learning of prototypes reflecting different allocation policies are being implemented and in a testing stage by means of a pattern partitioning scheme with a set-training regime as explained in previous sections.

7. EXPERIMENTS DESCRIPTION

Our initial experiments in parellelising neural nets were divided into two stages;

1. To establish if strong correlation exists between data dependencies and goodness of results expressed as the recall and generalisation abilities.

2. To compare goodness of results and speedup for diverse sizes of the training set.

In both analysis, to compare results, the neural net was trained sequentially and in parallel. Under pattern partitioning, the pattern space was divided into three sets and each set was assigned to one processor. Many series were performed and each one consisted of many runs for η values of 0.05 and 0.1. The following relevant performance variables were examined:

T: is the running time of the learning algorithm.
R = *r/Size*. Is the recall abitily of the neural net.
Were *Size* is the size of the submitted training set and *r* is the total number of patterns recognised when only those patterns of the training set are presented, after learning, to the network.
G = *g/S*. Is the combined recall and generalization ability.
Were *S* is the size of the total pattern space and *g* is the total number of patterns recognised when all the patterns are presented, after learning, to the network.
Sp $=T_{seq} / T_{par}$ is the ratio between the sequential learning and the parallel learning times.

$Rec_{B/C}$ = $Sp/(R_{seq} - R_{par})$ is the benefit-cost ratio for recall. It indicates the benefit of speeding up the learning process which is paid by the cost of loosing recall ability.
$Gen_{B/C}$ = $Sp/(G_{seq} - G_{par})$ is the benefit-cost ratio for generalization. It indicates the benefit of speeding up the learning process which is paid by the cost of loosing generalization ability.

Partitioning schemes were applied as follows (See figures 4.a and 4.b).
In the *first analysis stage*, for sequential backpropagation (SBP) the training set T_s was built by uniform selection of 30% of the pattern space *S*. As we decided to dedicate only three separate processors, for parallel training (PBP) three random disjoint subsets were selected according to the following criteria:

- One third of T_s for each subset. (Experiment PBP-10-D)
- 10% of *S* for each subset. (Experiment PBP-10-I)

In the *second analysis stage*, for sequential backpropagation (SBP) the training set T_s was built by uniform selection of X% of the pattern space *S*. For parallel training (PBP) three disjoint subsets of X/3% of *S* were selected. X was chosen as 30, 45 and 60. (Experiments PBP-X/3).

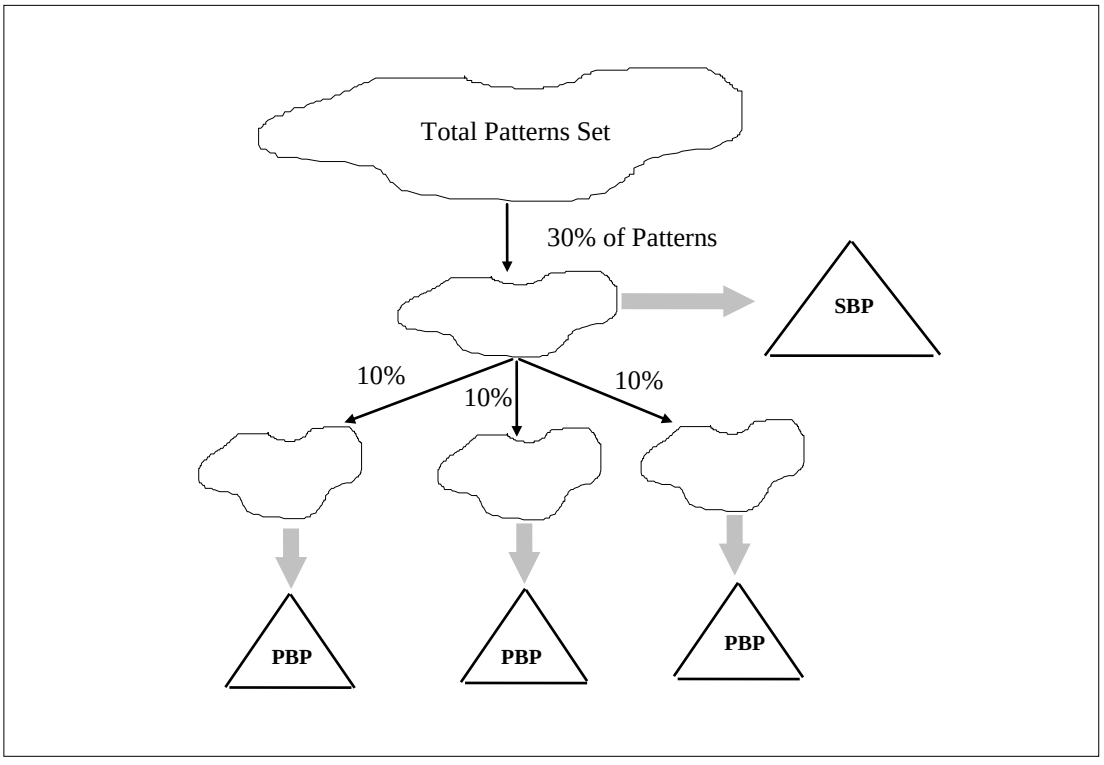

Fig. 4.a - Dependent-Data Partitioning Approach

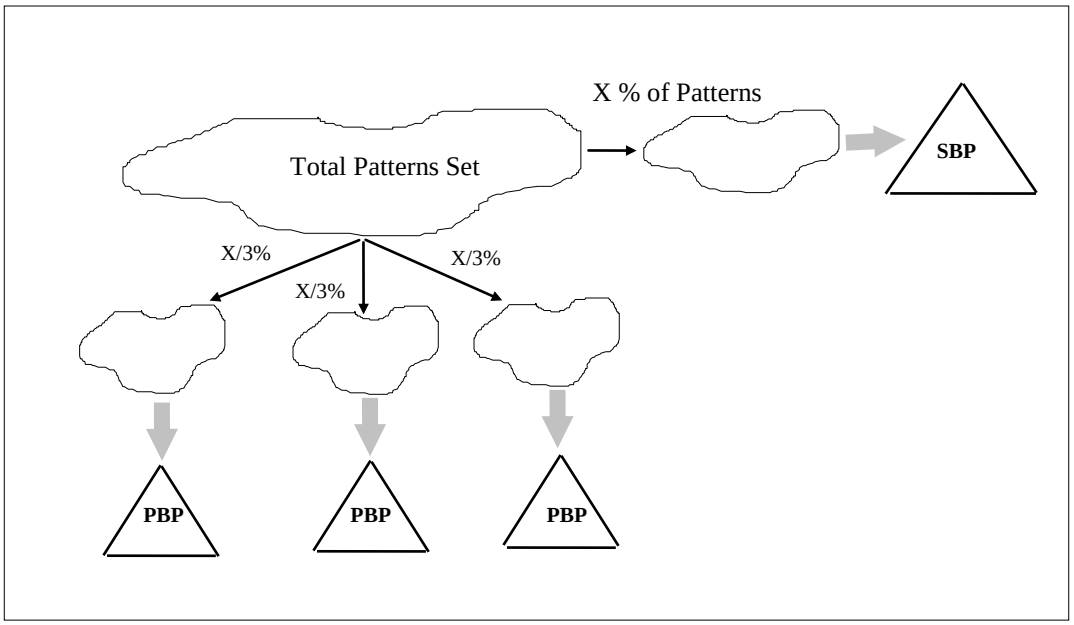

Fig. 4.b - Independent-Data Partitioning Approach

8. RESULTS

We show now some results from both investigation stages.

8.1 Stage I

To determine the possible existence of a strong correlation between data dependencies and goodness of results, experiments SBP, PBP-10-D and PBP-10-I were conducted. The issues studied through a large number of runs were; learning time, recall, generalization and number of iterations needed to reach an acceptable error value while training. The following figures and tables show the corresponding mean values.

Experiment	Learning Time	Recall	Generalization
SBP	1850	100	97
PBP-10-D	208.40	94.32	91.44
PBP-10-I	160.78	94.37	91.65

Table 1. Summary of T, R and G results in Stage I

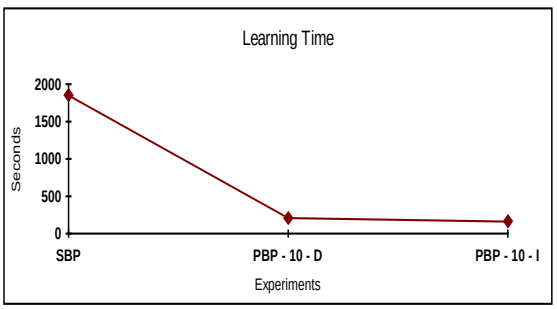

Fig. 5 - Values of T for Stage I under sequential and parallel processing.

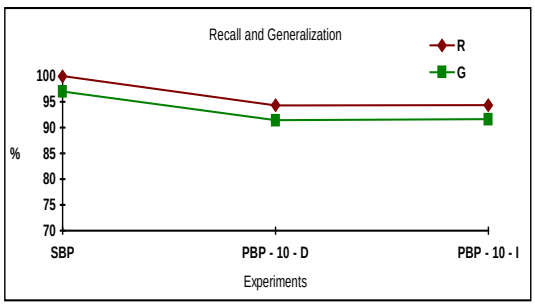

Fig. 6 - Values of R and G for Stage I under sequential and parallel processing.

As we can observe in the above table and figures important learning time reduction is achieved without significant loss of recall and generalization capabilities of the neural network. This results are attained by using either parallel partitioning approach. However the independent data criterion (PBP-10-I) seems to provide better performance values, possibly due to a more uniform distribution of data gathered.

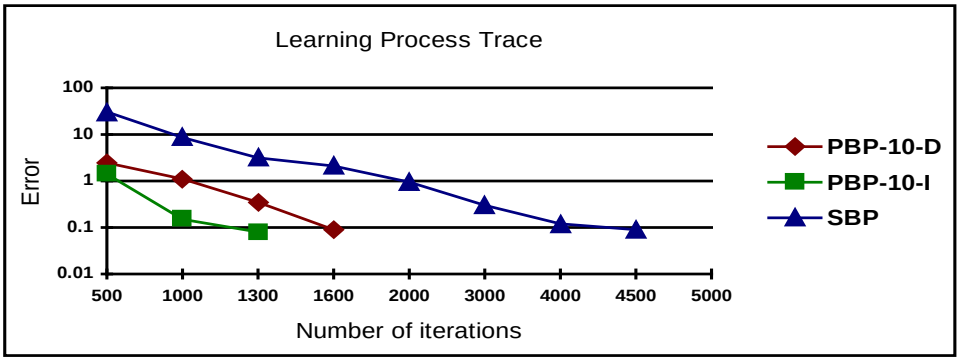

Fig. 7 Progress of the learning process under sequential and parallel processing

As it is shown in figure 7, the substantial time reduction obtained under either parallel approach arise from a lesser number of iterations to arrive to an acceptable error value. When the learning process is performed in parallel, each replication of the neural net is trained with fewer number of patterns and it learns not only due to its individual work but also for the “experience” shared with other replicas.

8.2 Stage II

To compare goodness of results and speedup for diverse sizes of the training set many tests were performed in experiments SBP-X and SBP-X/3 addressed to learning time, recall, generalization, speed-up, and benefit-cost ratio. Here, parallel processing was implemented under the independent data approach.

Experiment	Learning time	Recall	Generalization
SBP-30	1850	100	97
PBP-10	160.78	94.37	91.65
SBP-45	2700	100	98
PBP-15	434.75	96.3	94.47
SBP-60	4268	100	99
PBP-20	935.44	97.12	96.89

Table 2. Summary of T, R and G results in Stage II

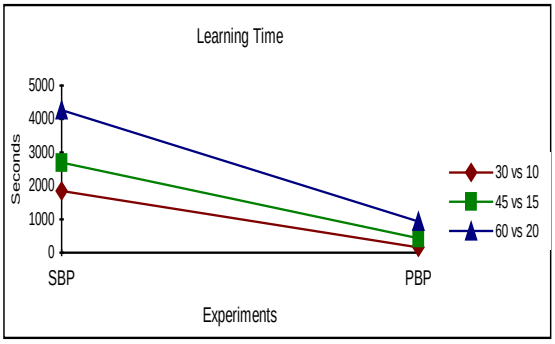

Fig. 8 - Values of T for Stage II, under sequential and parallel processing

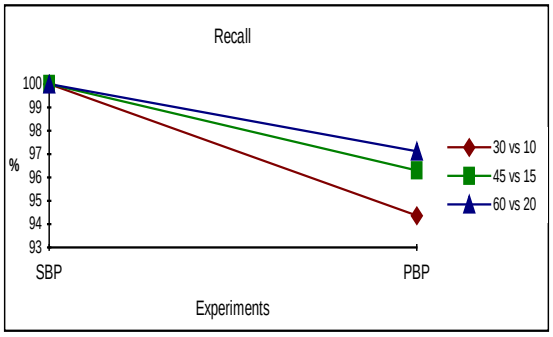

Fig. 9 - Values of R for Stage II under sequential and parallel processing.

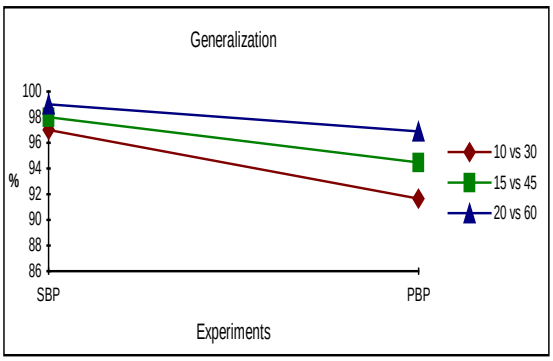

Fig. 10 - Values of G for Stage II under sequential and parallel processing.

As we can observe in Table 2 and Figure 8 considerable learning time reduction is achieved also for diverse sizes of the training set throughout experiments of Stage II.This results are attained by using the independet data approach.
Figures 9 and 10 show the associated loss in recall and generalization of the neural network for different sizes of the training set.
As anticipated, the amount of capabiliy detriment decreases as the training set size is incremented, and its values range from 2.9% to 5.6% for recall and from 2.1% to 5.3% for generalization.

Speed-up		
SBP-30 vs. PBP-10	SBP-45 vs. PBP-15	SBP-60 vs. PBP-20
11.51	6.21	4.56

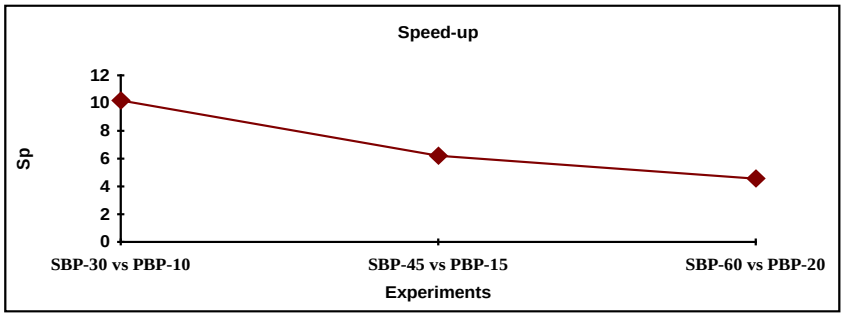

Fig. 11 - Speed-up values achieved through parallel processing.

The above table and figure indicate the Speed-up attained through parallel processing when different sizes of the portions of the pattern space S are selected for training the neural network. As the size of T_s increases then an effect of slowing the Speed-up arises but this consequence is compensated by an improvement in quality of results.

Benefit/Cost Ratio	SBP-30 vs. PBP-10	SBP-45 vs. PBP-15	SBP-60 vs. PBP-20
Recall B/C	3.836	1.678	1.583
Generalization B/C	2.994	1.779	1.857

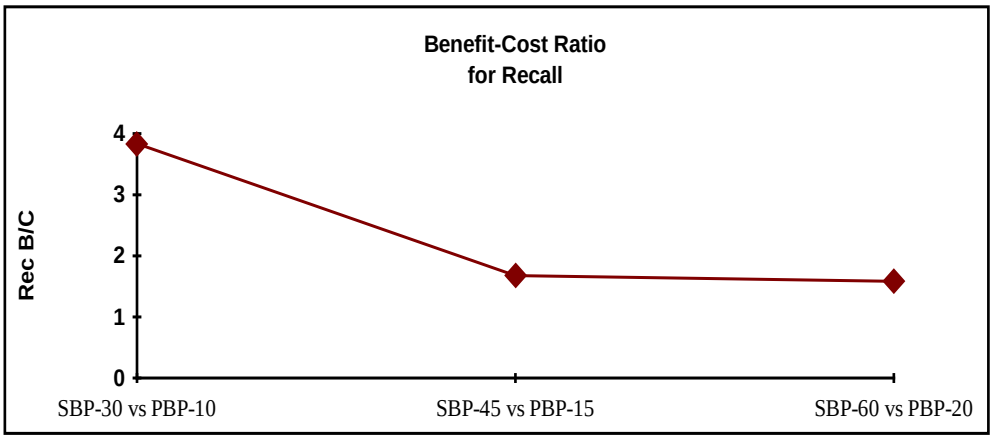

Fig. 12 - Benefit-Cost Ratio for Recall through parallel processing.

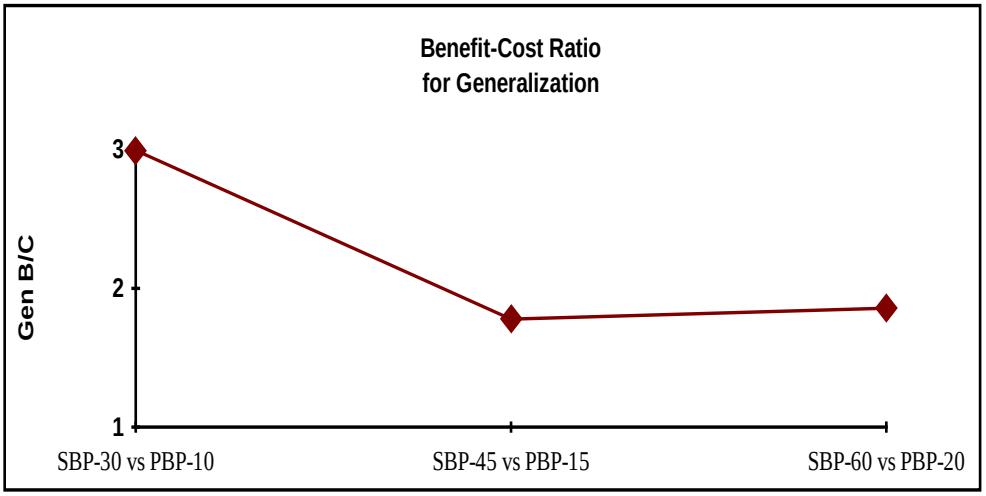

Fig. 13 - Benefit-Cost Ratio for Generalization through parallel processing.

The Benefit-Cost Ratio gives an indication of a benefit (speed-up) obtained by paying a cost (detriment in recall or generalization). This performance variable could be of great help when designing a parallel approach for training neural nets. It is clear that when larger sizes for the training set are used the benefit decreases and consequently the cost decreases. In any case, decision will depend on application requirements.

9. CONCLUSIONS

The time that it takes to train a neural network has long been an issue of research. The length of training time depends, essentially, upon the number of iterations required. This number depends on several interrelated factors. Some of them are; size and topology of the network, initialisation of weights and the amount of training data used.
A means of training acceleration based in the last mentioned factor is a parallel approach known as pattern partitioning.

In this paper we presented a feasible architecture for a system supporting parallel learning of backpropagation neural networks using a pattern partitioning scheme with a set-training regime. A preliminary set of experiments in our investigation, revealed that the beneficial effects of parallel processing can be achieved with minor capability loss. For the small neural net selected for the testing case a substantial acceleration ranging from 4 to more than ten times was attained by using as few as three processors.
As the distributed approach to parallelise the learning process showed its effectiveness, at the present time, tests with larger number of processors, different training set sizes and variable communication intervals are being performed for different neural networks.

9. ACKNOWLEDGEMENTS

We acknowledge the cooperation of the project group for providing new ideas and constructive criticisms. Also to the Universidad Nacional de San Luis and the CONICET from which we receive continuous support.

10. REFERENCES

[1] Berman F., Snyder L. - *On mapping parallel algorithms into parallel architectures*- Parallel and Distributed Computing, pp 439-458, 1987.
[2] Cena M., Crespo M. L., Gallard R. *Transparent Remote Execution in LAHNOS by Means of a Neural Network Device*. Operating System Reviews, Vol. 29, Nro 1, ACM Press, 1995.
[3] Colouris G., Dollimore J., Kindberg T. -*Distributed Systems: Concept and Design* - Addison-Wesley, 1994.

[1] Comer, D. E., Stevens, D. L. - *Internetworking with TCP/IP* - Vol. III - Prentice Hall.
[1] Foster, Ian T. : *Designing and Building Parallel Programs* - Addison Wesley, 1995.
[2] Freeman, J., Skapura, D. *Neural Networks. Algorithms, Applications and Programming Techniques*. Addison-Wesley, Reading, MA, 1991.
[3] Girau, B. - *Mapping Neural Network Back-Propagation onto Parallel Computers with Computation/Communication Overlapping*.
[4] Kumar, V., Shekhar, S., Amin, M.- *A Scalable Parallel Formulation of the Backpropagation Algorithm for Hypercubes and Related Architectures*. IEEE transactions on Parallel and Distributed Systems, Vol. 5. Nro.10, October 1994. pp 1073 - 1090.
[5] McEntire, P. L., O'Reilly, J. G., Larson, R. E. (Editors) : *Distributed Computing: Concepts and Implementations* - Addison Wesley, 1984 .
[6] Petrowski, A., Dreyfus, G., Girault, C.- *Performance Analysis of a Pipelined Backpropagation Parallel Algorithm*. IEEE. Trasactions Networks, Vol. 4, November 1993. pp 970 - 981.
[7] Plaut, D., Nowlan, S., Hinton, G. *Experiments on Learning by Backpropagation*. Tech. Report, CMU-CS-86-126, Carniege Mellon University, Pittsburg, PA, 1986.
[8] Rumelhart, D., Hinton, G., Willams, R. *Learning Internal Representations by Error Propagation.* MIT Press, Cambridge, MA, 1986.
[9] Rumelhart, D., McClelland, J. *Parallel Distributed Processing, vol. 1 y 2*. MIT Press, Cambridge, MA, 1986.
[10] Stevens, R.W.- *Advanced Programming in the UNIX Envionment*- AdDison-Weslwy Publishing Company. 1992.
[11] Stevens, R.W.- *UNIX Network Programing.* Prentice Hall-Englewood Cliff. 1990.